ODE: Ontology-based software Development Environment

Ricardo de Almeida Falbo, Ana Candida Cruz Natali,
Paula Gomes Mian, Gleidson Bertollo, Fabiano Borges Ruy
Computer Science Department, Federal University of Espírito Santo
Fernando Ferrari Avenue, CEP 29060-900, Vitória - ES - Brazil
{falbo, anatali, pgmian, gbertollo, fruy}@inf.ufes.br

Abstract

Software tools processing partially common set of data should share an understanding of what these data mean. Since ontologies have been used to express formally a shared understanding of information, we argue that they can be used to improve integration in Software Engineering Environments (SEE). In this paper we discuss an ontology-based approach to improve tool integration and present ODE, an ontology-based SEE.

Keywords: Software Engineering Environment, Software Process, Ontology, Knowledge Management.

1. INTRODUCTION

The demands on software development are widely increasing. At the same time, software applications are more complex, time to market is shortened, and the need to produce software at reasonable cost is great. In this context, better quality and productivity are critical factors for software development. To deal with these problems, it becomes essential to provide computer-based tools to support software engineers to perform their tasks. To be effective, however, these tools must work together.

Integration – of tools, processes, artifacts, and views – has been considered one of the most challenging issues on software engineering environment (SEE) research [1]. Integration demands consistent representations of software engineering information, standardized interfaces between tools, homogeneous means of communication between software engineers and tools, and an effective approach that enables SEE to move among various platforms [2].

It is necessary to build an infrastructure for tool integration. This infrastructure should be based on robust conceptual models. Software tools processing partially common set of data must share an understanding of what the data mean. We argue that ontologies are a promising means to achieve these conceptual models, since they can be used to promote common understanding, and they can be used as basis for comprehensive information representation and communication. In this way, an ontology-based approach can be used to improve tool integration in SEEs.

In this paper we present ODE (Ontology-based software Development Environment), a SEE developed using ontologies. In section 2 we discuss some issues concerning tool integration in SEEs and how ontologies can be used to improve integration. Section 3 presents ODE and its ontology-based approach for tool integration. This approach was applied in process and knowledge integration, which are discussed in sections 4 and 5, respectively. Section 6 discusses related works. Finally, in section 7, we report our conclusions.

2. ONTOLOGIES AND TOOL INTEGRATION IN SEE

Although benefits can be derived from individual CASE tools addressing separate software engineering activities, the real power of CASE (Computer-Aided Software Engineering) can be achieved only through integration [2]. The identification of the need for integrated support for these activities throughout the software lifecycle represents the genesis of Software Engineering Environments (SEEs) [1]. Thus, SEEs can be defined as integrated collections of tools that facilitate software engineering activities across the software lifecycle [1]. A SEE provides a means to

integrate people in a software development organization with the development process and with the supporting technology [3]. To do that, SEEs must provide an infrastructure for tool integration that should address five main issues [4]:

- data integration: the way tools share data;
- process integration: linkage between the tools and the software development process;
- control integration: the ability for one tool to notify and initiate actions in another;
- presentation integration: commonality of user interface;
- platform integration: the ability of tools to interoperate on a heterogeneous network.

But knowledge management (KM) can also be used to support developers during the software process. KM combines tools and technologies to provide support to the capture, access, reuse and dissemination of knowledge, generating benefits for the organization and their members [5]. Using a KM approach, knowledge created during software process can be captured, stored, disseminated, and reused, so that better quality and productivity can be achieved. KM can be used to better support management activities, such as software process definition, people allocation and estimation, software construction activities, such as requirement analysis and test case design, and quality assurance activities, such quality planning and control. Consequently, SEEs and knowledge management complements each other in supporting developers during the software process to produce better quality software [5]. In this context, we should consider another dimension in tool integration: knowledge integration.

It is necessary to build an infrastructure for tool integration that considers all these six dimensions. This infrastructure should be based on robust conceptual models. Software tools processing partially common set of data must share an understanding of what the data mean. Since ontologies are a promising means to achieve these conceptual models, an ontology-based approach can be used to improve tool integration in SEEs.

According to Guarino [6], "an ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world". An ontology consists of concepts and relations, and their definitions, properties and constrains expressed as axioms [7].

An ontology is a representation vocabular*y,* often specialized to some domain or subject matte*r.* More precisely, it is not the vocabulary as such that qualifies as an ontology, but the conceptualizations that the terms in the vocabulary are intended to capture [8]. Ontologies are quintessentially content theories, because their main contribution is to identify specific classes of objects and relations that exist in some domain. Without ontologies, or the conceptualizations that underlie knowledge, there cannot be a vocabulary for representing knowledge [8].

Common ontologies are used to describe ontological commitments for a set of agents so that they can communicate about a domain of discourse without necessarily operating on a globally shared theory. Ontological commitments are agreements to use a shared vocabulary in a coherent manner, and an ontology should be designed to anticipate the uses of this vocabulary. In other words, one should be able to define new terms for specific purposes based on the existing vocabulary (extendibility) [9]. Also, an ontology should require only the minimal ontological commitment sufficient to support the intended knowledge sharing activities (minimal ontological commitment). It should make as few claims as possible about the world being modeled, allowing the parties committed to the ontology freedom to specialize and instantiate the ontology as needed [9].

Since an ontology does not intend to describe all the knowledge involved in a domain, but only that one that is essential to conceptualize the domain (minimal ontological commitment [9]), ontologies can be used in SEEs as coarse-grained models that can be enriched when necessary. Moreover, an ontology can be developed without commitment to a specific formalism (minimal encoding bias [9]). Several approaches can be used to implement it using different technologies. For

example, we have used objects to implement frameworks, and hypertexts to implement tutorials, both based on the same ontology [10].

If the tools in a SEE are built based on ontologies, tool integration can be improved. The same ontology can be used for building different tools supporting correlated software engineering activities. Moreover, if the ontologies are integrated, integration of tools built based on them can be highly facilitated. Also, ontologies are particularly important for Knowledge Management. They constitute the glue that binds KM activities together, allowing a content-oriented view of KM. Ontologies define the shared vocabulary used in the KM system to facilitate communication, integration, search, storage and representation of knowledge [11]. Thus, adopting an ontology-based approach to tool integration, we believe that we can advance towards Semantic SEEs.

A Semantic SEE can be viewed as a SEE in which part of the information handled has a formal meaning (semantics) associated, augmenting its tools' ability to work in cooperation each other and with human developers. Tools committed them self with an ontology can share knowledge, since the ontology defines the common meaning.

The term "Semantic SEE" was coined using an analogy with Semantic Web [12]. Semantic Web aims to organize Web information, adding meaning to them, and allowing machines to process and analyze Web contents. The main goal of a Semantic SEE is analogous: to organize software engineering information, adding meaning to them, and allowing tools to share information. In a Semantic SEE, software engineering knowledge is accessible not only to human developers, but also to automated tools. Adapting the discourse of Bechhofer et al. [12], the key idea is to have software engineering data on the SEE defined and linked in such a way that its meaning is explicitly interpretable by software tools rather than just being implicitly interpretable by human developers.

3. ODE – AN ONTOLOGY-BASED SEE

Since ontologies have been used to express formally a shared understanding of information, we advocate their use to go ahead Semantic SEEs. Ontologies should be developed to address software engineering sub-domains, such as software process, software quality, risk analysis, and so on. A systematic approach for building ontologies, such that described in [13] [14], should be applied.

It is worthwhile to point that ontologies are at knowledge level [6]. Ontologies can be view as domain models in a domain engineering approach [14]. Based on an ontology, during the infrastructure specification phase of domain engineering [15], domain infrastructures can be developed. Several infrastructures can be developed for the same sub-domain, each one proper to some development technology.

Since we are most interested in the object technology to build an infrastructure for integrating tools, we applied the approach for deriving object frameworks from domain ontologies described in [13] [14]. These frameworks are used as the basis to build and integrate tools in the SEE. For each tool or service to be developed, a requirement specification must be done in a with-reuse approach, i.e., using and specializing the frameworks derived.

This is the approach used in ODE (**O**ntology-based software **D**evelopment **E**nvironment), a process-centered SEE. Since a SEE can be viewed as a (partial) automation of the software process, ODE's process kernel is built based on a software process framework derived from a software process ontology [16]. Tools for process definition and project tracking are also built based on this framework. Moreover, all ODE's internal tools agree on this ontology.

To address software quality control in ODE, we adopted the same strategy. We developed a software quality ontology (see [14]), and using this ontology, we built tools for quality management, such as ControlQ [5], a tool that supports product quality planning and evaluation. To deal with risk analysis in ODE, the same approach is being used.

ODE's architectural style reflects its basis on ontologies. It has two levels, as shown in figure1. The base or application level concerns application classes, which model the objects that address some software engineering activity. The meta-level (or knowledge level) defines classes that describe knowledge about objects in the base level.

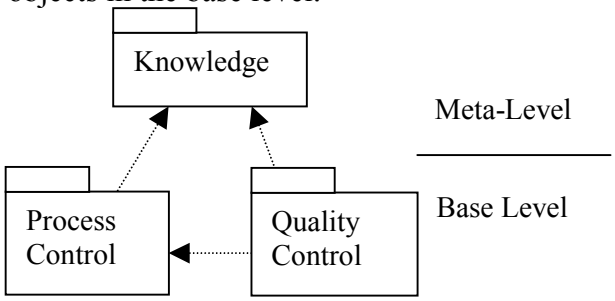

Figure 1. ODE's two-layered architecture.

The classes in the meta-level are derived directly from the ontologies, using the systematic approach to derive object frameworks from ontologies described in [13] [14]. All classes derived directly from the ontology are prefixed by the character "K", indicating that they constitute the knowledge in ODE. Meta-level objects can be viewed as items of an ontology instantiantion.

The classes in the base level are also built based on the ontologies. The main classes and associations are derived from the ontology, preserving the same constraints as Knowledge's model. Also several classes in the base level have a corresponding Knowledge class in the Knowledge package. In this way, the meta-level can be used to describe base-level objects' characteristics. However, since an ontology does not intend to describe all the knowledge involved in a domain, but only that one that is essential to conceptualize the domain (minimal ontological commitment [9]), new classes, associations, attributes and operations are defined to deal with specific design decisions made in the application level. In fact, the ontology is a general, common sense model, and then it does not contain all necessary modeling elements to treat applications' requirements. Ontologies are also used to deal with knowledge management in ODE. Next, we discuss the application of our ontology-based approach to process and knowledge integration.

4. PROCESS INTEGRATION IN ODE

The explicit representation of processes, their activities, products and interactions, is the foundation on which modern integrated development environments are built. In providing more powerful ways of describing and implementing software development processes, Process-centered Software Engineering Environments (PSEE) have also provided powerful means of integrating processes and tools, and (partially) automating tasks [1]. Based on that, ODE's process kernel was built using the software process ontology presented in [16] as its foundation. This ontology was designed to support software process definition, tracking and integration in a PSEE. Every tool compromised with this ontology will share a common vocabulary, facilitating the communication between tools and allowing reuse.

The concept of activity is in the core of any software process model. Activities can occur in several levels, from an elementary task to a development process phase. An activity is the basic transformational action primitive that uses input artifacts to produce output artifacts, supported by resources. In the definition of a software process, it is important to determine how the activities will be performed. To do that, we must establish procedures to be adopted in the accomplishment of the activities. Software processes are defined based on a paradigm. Also, life cycle models are used as a reference to guide the definition of activities. Figure 2 shows part of the software process ontology, written in LINGO [7], which semantics is presented in Figure 3, with the whole-part axioms.

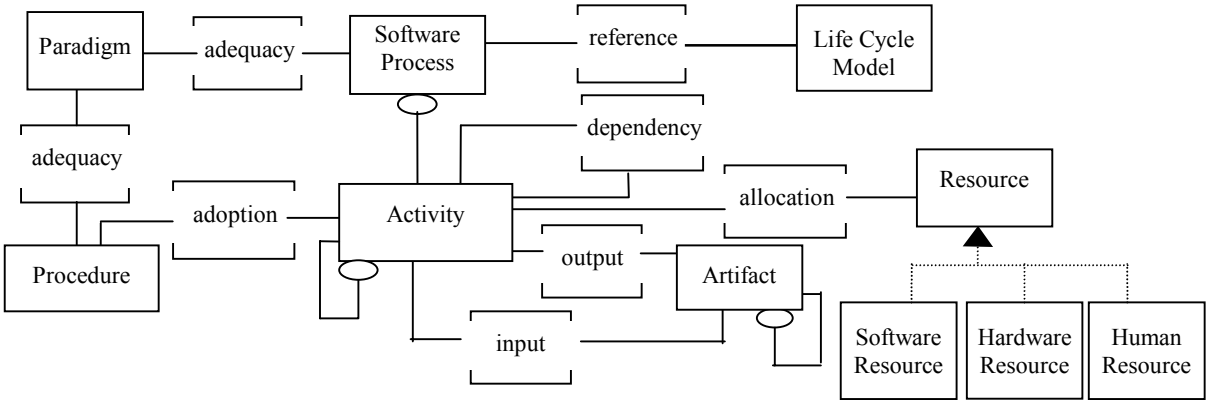

Figure 2. Software process ontology [16].

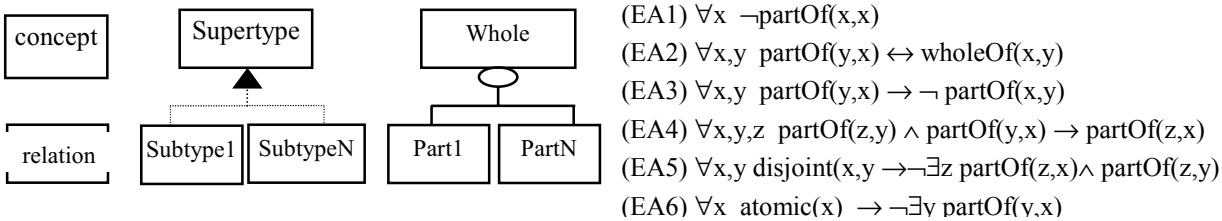

Figure 3. LINGO's notation [7].

Several axioms were defined in this ontology. These axioms were developed to provide a basic interpretation of the concepts in the ontology and to capture and write down the constraints imposed by the domain [7]. The following axiom, for instance, defines the concept of *pre-activity* from the input and output relations: $(\forall a_1,a_2)(\exists s)(input(s,a_2) \wedge output(s,a_1)) \rightarrow preactivity(a_1,a_2)$.

Based on the software process ontology, a Java framework was developed, using the approach described in [13] [14]. This framework is part of the Knowledge package (Figure 1). Figure 4 shows part of the Knowledge package's class diagram derived from the software process ontology. This ontology-to-object framework derivation process is shown in details in [13].

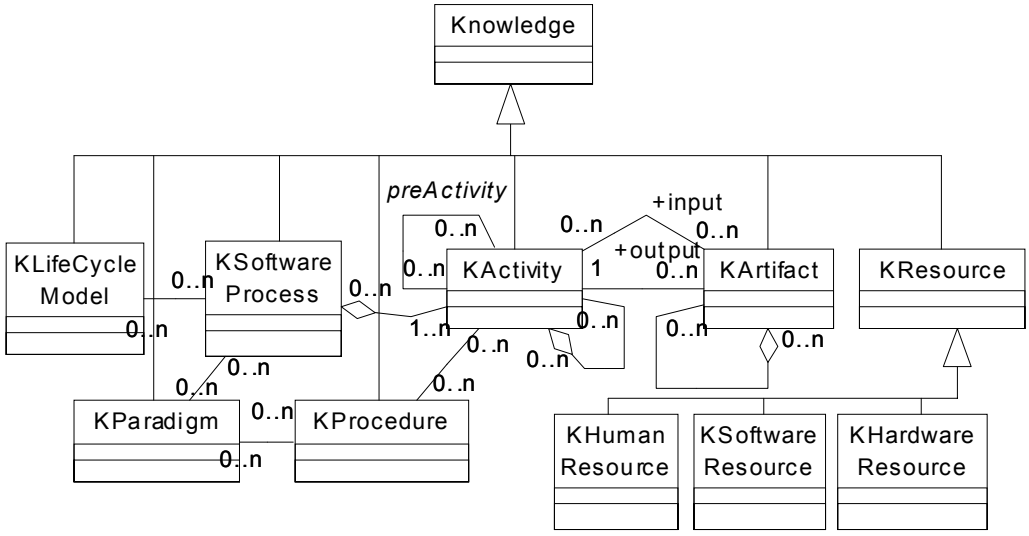

Figure 4. Software Process Knowledge Package.

Also based on the ontology, but adding new features to satisfy process definition and project tracking requirements, we developed the Process Control package. Figure 5 shows partially the Process Control package's class diagram. This package defines the classes that are used to promote

process integration in ODE. The relationships between meta-level (knowledge) objects and base level (process control) objects are shown as attributes in order to distinguish from other kinds of associations. In this way, we define in the Knowledge package the general knowledge about software processes and use this knowledge to guide actual software process definition.

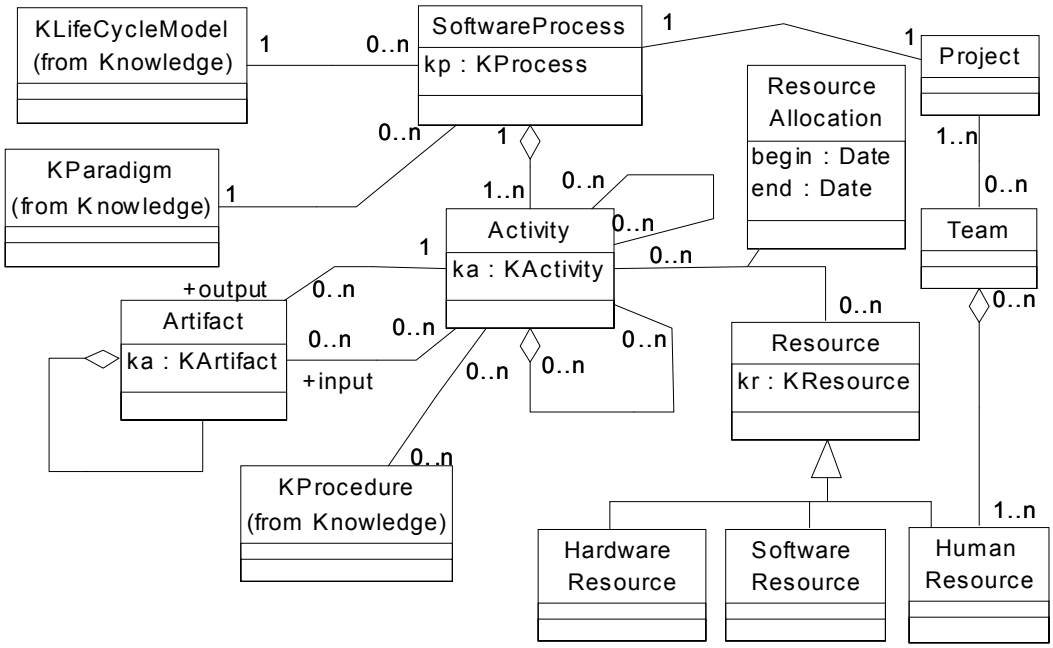

Figure 5. Process Control Package.

To illustrate how ODE uses knowledge, consider the following situation. We know that, in general, a design activity can be decomposed into architectural design and detail design activities. This general knowledge is described in the Knowledge package. In a software process definition, ODE gives this guideline to the project manager, who is free to accept or reject the suggestion. If it is accepted, two new activities are created in the Process Control package, with the corresponding aggregation associations. Figure 6 shows this example in the context of ODE's software process definition tool. Once defined, the process can be executed and tracked. ODE has several tools to support construction activities (such as CASE tools for object-oriented analysis and design), management activities (such as tools to support point-function analysis and risk management) and quality control activities (such as a tool for quality planning and control). For project tracking, ODE offers a tool that presents the process and the state of its activities (done, awaiting, executing, and so on), as shown in Figure 7.

It is worthwhile to show how process integration in ODE was facilitated, using an example. At Knowledge Package, there is an instance of KSoftwareResource describing that a quality management tool can be allocated to a planning activity. This information is useful during software process definition. In ODE, there is a quality management tool, named ControlQ, that is an instance of Software Resource. If ControlQ is allocated to the planning activity of a given project, it can be used when this activity is in execution. In this way, process integration is realized. Since a software process is established, it can be controlled. To support process control, ODE's tool menu is configurable. In the example above, if the project manager X (a human resource) is allocated to the planning activity and ControlQ is also allocated to this activity, ControlQ is shown in the tool menu of ODE, when this project manager logs on the environment.

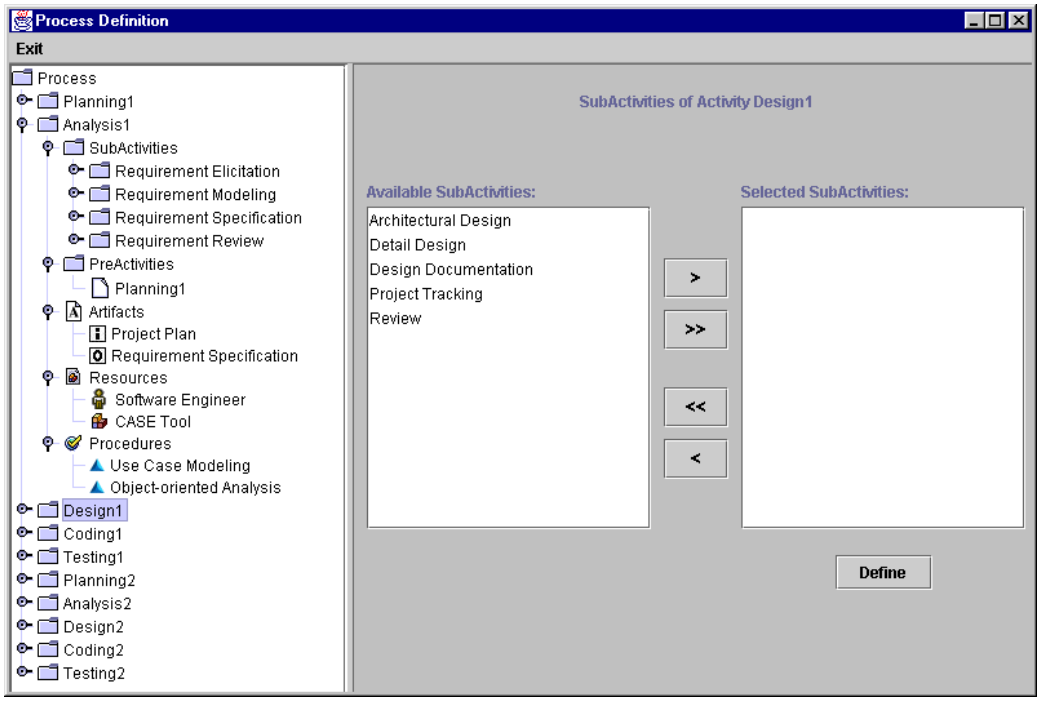

Figure 6. ODE's process definition tool.

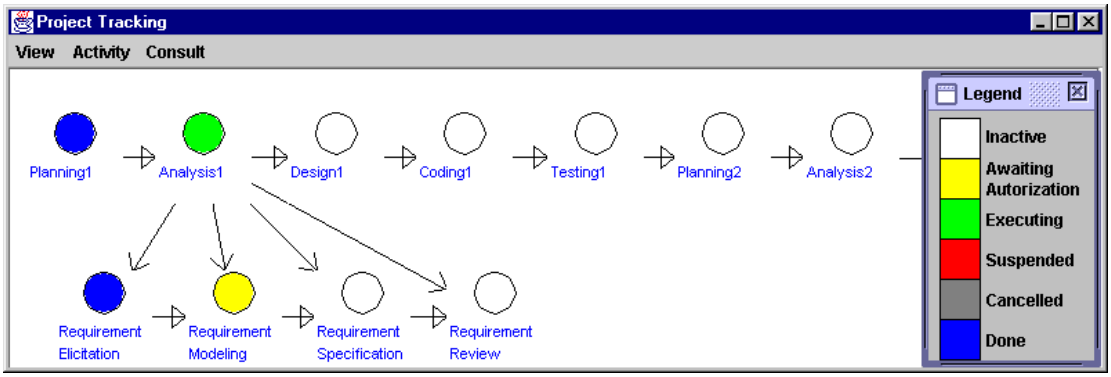

Figure 7. ODE's project tracking tool.

5. KNOWLEDGE INTEGRATION IN ODE

As pointed in section 3, the Knowledge package contains all the classes derived directly from the ontologies. In this package, there are the two most important ontologies to ODE: the software process ontology [16] and the software metrics ontology [14]. These ontologies are important because they are used as the foundation to the environment. But other ontologies can be developed in ODE, using ODE's ontology editor, ODEd [10].

ODEd supports the definition of concepts and relations, using graphic representations, and promotes automatic generation of some classes of axioms. Also, ODEd supports the derivation of object-oriented frameworks from ontologies, and an ontology-based knowledge acquisition approach. Based on the ontology developed, classes and databases are automatically generated by ODEd. Once the database is created, it is possible to instantiate the ontology. To do that, customized windows are generated, based on the ontology contents, to allow the instance data input [10].

Therefore, besides the basic ontologies of ODE, domain ontologies can be developed. From them, domain knowledge can be captured and stored in ODE's knowledge repositories. The Knowledge package, thus, contains software engineering and domain knowledge (figure 8).

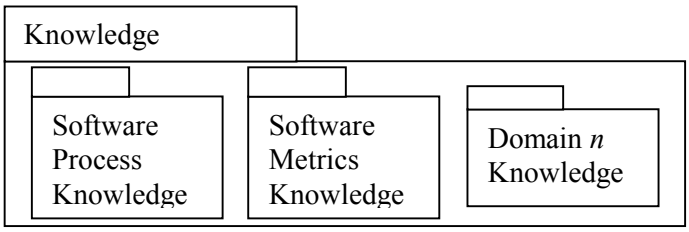

Figure 8. The Knowledge Package.

But to effectively integrate knowledge, it is necessary to provide knowledge management services. In ODE, an infrastructure for knowledge management (KM) was provided [5].

The organizational memory (OM) is at the core of this infrastructure, supporting knowledge sharing and reuse. Arranged around the OM, knowledge management services actively provide useful information to users working on knowledge-intensive tasks. These knowledge management services include creation, capture, retrieval, access, dissemination, use, and preservation of the organization's knowledge, as shown in Figure 9.

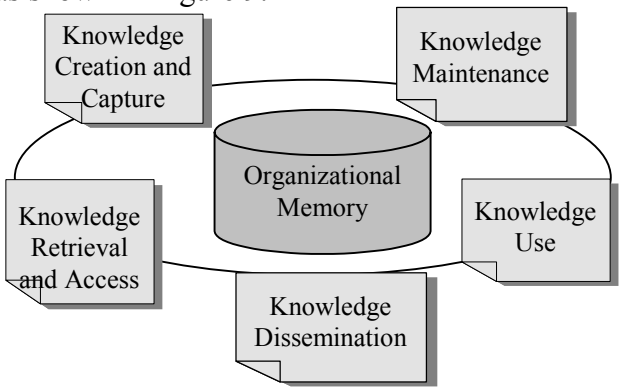

Figure 9. Knowledge management infrastructure.

ODE's organizational memory contains three types of knowledge itens: artifacts, instances of ontologies and lessons learned. Artifacts and instances of ontologies correspond to the formal knowledge. Lessons learned are the informal knowledge. The OM holds information from previous projects so that users can use them to solve similar problems and to perform similar tasks.

Arranged around the OM, there are the KM services. These services are grouped in two categories: general services, which are actually incorporated to ODE, and tool specific services, which cannot be made available to the environment, because need to be customized to a specific tool. General services include:

- *Knowledge Capture*: Since ODE deals with three kinds of knowledge, it must offer facilities to capture each one of these type. When dealing with lessons learned, generally, project-level knowledge must be handled to become an organizational knowledge. In ODE, there is a tool supporting a workflow for approving a lesson learned. First, a developer inputs a lesson learned in the OM. At this moment, this knowledge is not available to other developers. The knowledge manager must evaluate and adapt the lesson learned so that it can be considered knowledge at the organizational level. Once approved, the lesson learned is made available. Artifacts created during the software process are also available as knowledge items. In the current stage, the ODE's configuration management system is the base for dealing with artifacts as knowledge items. Finally, the knowledge manager is responsible for creating the instances of the ontologies that are useful to the organization. In ODE, for each of the basic ontologies (software process and software metrics), there is a

tool supporting its instantiation. Domain ontologies instantiation can be done using ODEd, as discussed above.

- *Knowledge Retrieval and Access*: Knowledge management in ODE supports information access through searching. An ODE user can search for any kind of knowledge in the OM: formal knowledge (artifacts and ontology instances) or informal (lessons learned).
- *Knowledge Use*: Once a knowledge item is selected for use, the user can identify what part he/she wants to use and a new knowledge item is created based on the previous one. Some reuse information is shown, including when and how often this item has been used and who used it. Finally, the user must evaluate the reused item to help knowledge maintenance. It includes evaluation information about if the item was useful, problems that appeared when reusing it, and solutions which have been applied.
- *Knowledge Maintenance*: For maintenance and evolution of the OM, it is necessary to take into account users' feedback. Based on the user feedback, the knowledge manager can decide what knowledge item is obsolete or which one had never been used. The knowledge manager can exclude knowledge items by himself or can require the support of a software agent. This software agent can be set to alert the knowledge manager to realize an OM's maintenance at defined time intervals or when the OM has reached a defined size. The software agent can also suggest some knowledge items to be excluded.

Knowledge dissemination is a tool specific service. While knowledge retrieval and access is a user-initiated search, knowledge dissemination is initiated by the system, without requiring the user to explicitly formulate a query. Knowledge dissemination is particularly important when users are not motivated to look for information or when they are not aware of the need for information in the first place. But it is not possible to provide proactive knowledge dissemination without knowing details about the task being done. Thus, knowledge dissemination is not provided by the environment. It is a service that must be implemented in each tool with KM support.

The dissemination service is carried through by software agents who monitor the users' actions as they work and inform them about potential relevant knowledge. Users can browse the various knowledge items and then select and reuse one of them. A general agent of ODE, named *ODE'sMonitorAgent*, monitors the users' actions and verifies when a tool with KM support starts to be used. When a tool with KM support is initiated, it is necessary that a specific agent for this tool starts to monitor users' actions for knowing *when* to present knowledge and *what* is considered relevant knowledge for that task.

Since only one software agent is not able to know how all ODE's internal tools work, each tool with KM support must have its own specific agent, defined and configured in tool's development time. Thus, it is not possible to adopt an unique structure for knowledge dissemination in ODE. The adopted solution was the development of a framework to facilitate the definition of this service in specific ODE's internal tools.

Figure 10 shows the framework developed to facilitate the implementation of knowledge dissemination in a specific tool. This tool must be a tool with KM support, and then its main application must inherit from the *KMSupportedTool* class. All instances of this class, that is KM supported tools, have access to some *knowledge repositories*. The tool's developer has to define, in development time, which knowledge repositories will be accessible by the tool.

Moreover, it is necessary to define the software agent that will monitor the tool user's actions. This software agent must be a specialization of the *ToolSpecificAgent* class. This class has two abstract methods *knowledgeRetrieval*() and *knowledgePresentation*(), that must be implemented by the concrete software agent classes, defining *what* to present (type of knowledge) and *when* to present the knowledge, respectively.

KM software agents are associated with the tool's main applications in which they will act, and are activated by the *ODE'sMonitorAgent* when the tool is initiated. Thus, the specific tool agent created is capable to retrieve and actively disseminate knowledge to the user of an ODE's tool at the moment of its use.

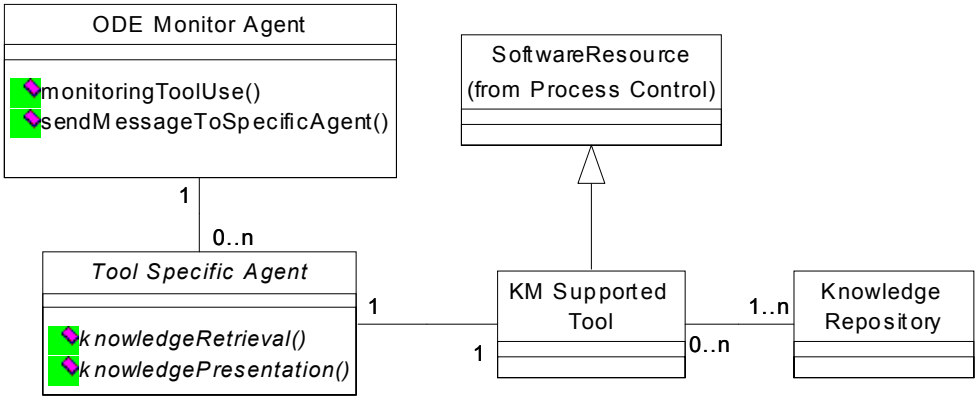

Figure 10. Framework for Knowledge Dissemination in ODE's Internal Tools.

This framework was used, for instance, to implement ControlQ [5], a tool that supports software quality planning and tracking. ControlQ's main application inherits from *KMSupportedTool* class, and has a *ControlQAgent* class associated, that inherits from *ToolSpecificAgent.* ControlQ's functionalities regarding quality planning allows to define quality evaluation activities that will be carried along the project. Quality planning in ControlQ involves the following steps:

1. Select a project to which a quality plan will be created and define the project's software artifacts that will be evaluated;
2. For each artifact, identify which quality characteristics will be used to evaluate it.
3. For each non measurable quality characteristic, define how it is decomposed into sub-characteristics;
4. Define how to measure the identified measurable characteristics, choosing adequate metrics, as shown in Figure 11.
5. Define quality evaluation activities, integrating them into the software process. For each artifact defined in step 1, a set of quality evaluation activities is defined.

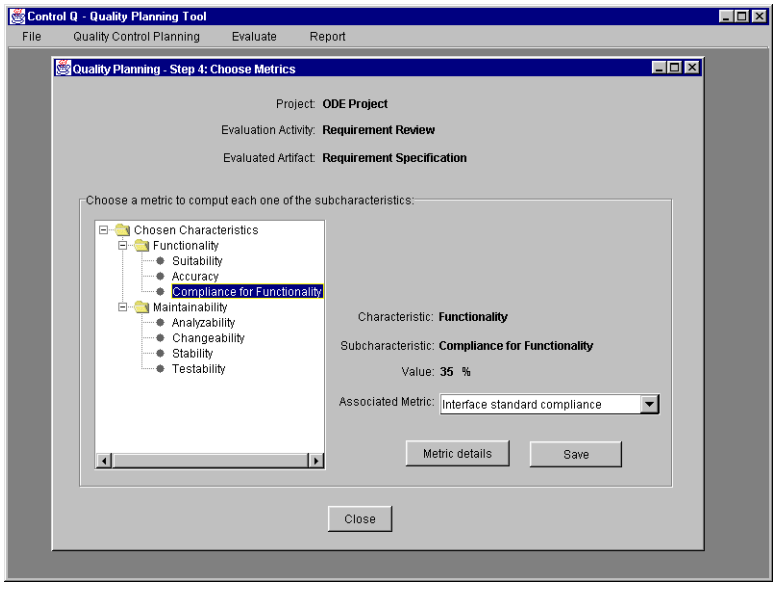

Figure 11. Choosing metrics in ControlQ.

In steps 2 to 4, ControlQ offers a knowledge dissemination service. ControlQ's software agent monitors users actions as they work in ControlQ. When the user is working in one of these steps, it acts, identifying user's knowledge needs, and retrieving past and similar experiences. These agents disseminate lessons learned that relate to success or failure. Also, they disseminate other quality control plans already defined and evaluated. So, based on similar experiences, users can make decisions based not only on their own knowledge but also on organizational knowledge.

It is worthwhile to emphasize the importance of the ontologies in this KM infrastructure. In the context of the knowledge management, ontologies define the shared vocabulary used in the KM system to facilitate communication, search, storage, and representation. Ontologies constitute the glue that binds knowledge subprocesses together. Ontologies open the way to move from a document-oriented view of KM to a content-oriented view, where knowledge items are interlinked, combined, and used [11]. In ODE's knowledge management approach, ontologies are used to structure the OM, as well as to support the main knowledge services, such as search and reuse of knowledge items. Figure 12 shows part of ODE's internal model that deals with KM.

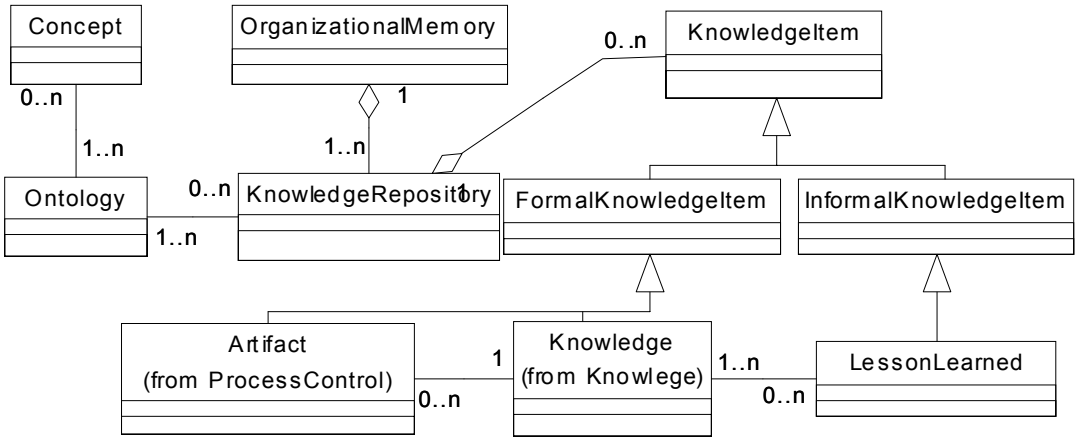

Figure 12. Part of the KM Package.

6. RELATED WORK

There are several SEEs and PSEEs described in the literature, such as Argo [17] and SPADE [18]. From the process integration point of view, practitioners' most important need is to describe processes with the purpose of understanding and communicating them. However, almost always, the languages used to describe processes in such environments are complex, extremely sophisticated, and strongly oriented towards detailed modeling of processes. Argo's process model, for example, has four modeling perspectives: global work flow view, local decision-oriented view, product view and organization view. SPADE, on the other hand, exploits Petri nets to model processes.

We believe that the ontological approach can be helpful, since a software process ontology does not intend to describe all the knowledge involved in the software process domain, but only that one that is essential to conceptualize the domain (minimal ontological commitments), as discussed in section 3. Moreover, we advocate that this approach is more widely applicable and can be used to improve tool integration in general. For example, we applied this approach to treat quality control in ODE [5].

Finally, this ontological approach is being used to make ODE a Domain-Oriented Design Environment, supporting knowledge management. Using a domain ontology to describe the domain knowledge and deriving a framework from it, we can promote domain knowledge sharing in a SEE. This is done in ODE through ODEd (ODE's ontology editor) [10].

7. CONCLUSIONS

In this paper, we presented an ontological approach to deal with process and knowledge integration in SEEs and how it is materialized in ODE. Using such approach we argue that we are going towards what we are calling Semantic SEE: a SEE which tools are developed based on ontologies. This makes easier knowledge management and process control.

Acknowledge

The authors thanks CAPES and CNPq for the financial support to this work.

REFERENCES

[1] W. Harrison, H. Ossher, P. Tarr, *Software Engineering Tools and Environments: A Roadmap*, in Proc. of The Future of Software Engineering, ICSE'2000, Limerick, Ireland, 2000.

[2] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 5th Edition, 2001.

[3] A.M. Christie, *Software Process Automation: The Technology and Its Adoption*, Springer-Verlag, 1995.

[4] S.L. Pfleeger, *Software Engineering: Theory and Practice*, 2nd edition, Prentice Hall, 2001.

[5] A.C.C. Natali, R.A. Falbo, *Knowledge Management in Software Engineering Environments*, Proc. of the 16th Brazilian Symposium on Software Engineering, Gramado, Brazil, 2002.

[6] N. Guarino, *Formal Ontology and Information Systems*, in Formal Ontologies in Information Systems, N. Guarino (Ed.), IOS Press, 1998.

[7] R.A. Falbo, C.S. Menezes and A.R.C. Rocha, *A Systematic Approach for Building Ontologies*, in Proc. of the IBERAMIA'98, Lisbon, Portugal, 1998.

[8] B. Chandrasekaran, J.R. Josephson, V. Richard Benjamins, *What Are Ontologies, and Why Do We Need Them?*, IEEE Intelligent Systems, January/February 1999.

[9] T. Gruber, *Towards principles for the design of ontologies used for knowledge sharing*, Int. J. Human-Computer Studies, 43(5/6), 1995.

[10] P.G. Mian, R.A. Falbo, *Supporting Ontology Development with ODEd*, Proceedings of the 2nd Iberoamerican Symposium on Software Engineering and Knowledge Engineering, Salvador, Brazil, 2002.

[11] S. Staab, R. Studer, H.P. Schurr, Y. Sure, *Knowledge Processes and Ontologies*, IEEE Intelligent Systems, January/February, Vol. 16, No. 1, 2001.

[12] S. Bechhofer, I. Horrocks, C. Goble, R. Stevens, *OilEd: a Reason-able Ontology Editor for the Semantic Web*, Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.

[13] G. Guizzardi, R.A. Falbo, J.G. Pereira Filho, *Using Objects and Patterns to Implement Domain Ontologies*, Journal of the Brazilian Computer Society, vol.1, n. 8, July 2002.

[14] R.A. Falbo, G. Guizzardi, K.C.Duarte, *An Ontological Approach to Domain Engineering*, Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, 2002.

[15] G. Arango, R. Prieto-Diaz, "Domain Analysis Concepts and Research Directions", in Domain Analysis and Software Systems Modeling, IEEE Computer Society Press, 1991.

[16] R.A. Falbo, C.S. Menezes, and A.R.C. Rocha, *Using Ontologies to Improve Knowledge Integration in Software Engineering Environments*, in Proceedings of SCI'98/ISAS'98, Orlando, USA, July, 1998.

[17] J. Lonchamp and F. Seguin, *The Argo Project*, in Proc. of the 9th Int. Conference on Software Engineering and Knowledge Engineering, SEKE'97, Spain, 1997.

[18] S. Bandinelli, E.Di Nitto, A. Fuggetta, *Supporting Cooperation in the SPADE-1 Environment*, IEEE Transactions on Software Engineering, vol. 21, no. 5, pp 440-453, May 1996.